Ashim Ghimire

DATA ENCRYPTION IN ANDROID

Thesis
CENTRIA UNIVERSITY OF APPLIED SCIENCES
Information Technology
October 2020

ABSTRACT

Centria University of Applied Sciences	Date	Author
	October 2020	Ashim Ghimire

Degree programme Information Technology	
Name of thesis DATA ENCRYPTION IN ANDROID.	
Instructor Bartosz Wieczorek	**Pages** 35 + 2
Supervisor Jari Isohanni	

The importance of data security is growing every year and the threats to it are increasing every year. Data security in mobile applications is never easy. If a mobile phone is stolen and the screen lock is breached, then the security of data is threatened. Raisoft which is a Finnish software company owns mobile application called RaiMobile. It is an application which can run with or without internet connection. RaiMobile is currently in development mode and its data security is to be improved.

The aim of this thesis is to improve the data security of the RaiMobile application. In this thesis a proof of concept application for RaiSecure is made. It tries to tackle the security problem faced by RaiMobile and implement a solution following the best security practices. The solution implemented in RaiSecure needs to be compatible with the RaiMobile. RaiSecure is a small application containing a SQLite database with dummy data and with three screens. The main task of RaiSecure is to keep the data in the database and local storage safe. The data in the database is kept safe by encrypting the database.
Since RaiSecure is an offline application key used for database encryption it is stored in a local data storage. The key is again used for decrypting database. The key is kept safe by encrypting it with key material from Android KeyStore provided by Android. The key is later saved in a local encrypted shared preference. RaiSecure tries to keep the data protected by using secure encryption and with the best security practices followed in Android Community.

Key words
Android KeyStore, Decryption, Encryption

ABBREVIATIONS

API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
OWASP	Open Web Application Security Project
REST	Representational State Transfer
UML	Unified Modeling Language
XML	Extensible MarkUp Language

CONTENTS

GRAPHS

FIGURES

TABLES

1 INTRODUCTION

“Oy Raisoft Ltd” was founded in 2000. Raisoft provides solutions for care planning, quality monitoring, and resource management in areas of elderly care, rehabilitation, and mental health” (Raisoft, 2020). The raiSecure Android application described in this thesis is a proof of a concept for a bigger Android application RaiMobile, which is a mobile version of a software provided by Raisoft. RaiMobile is a dual application, which supports offline/online mode. The user can use the application offline and is able to synchronize the data to the Raisoft server when the device is connected to the internet.

The aim of this project is to encrypt the data in RaiSecure in the best possible way which can later be used in RaiMobile. Customer data security is important for any company, any compromise to it can lead to a negative image of that company, business loss, penalty fine and even the collapse of the company. Since RaiSecure is an offline mobile application the data is stored in a mobile local storage and it is always at risk. The data stored in a database, a local file or shared preference are at high risk if the device is lost or has root privileges. Thus, encryption is the best way to protect the data from any attack. This solution will improve the way data is secured in RaiMobile. With this solution more layers of security are added to the application data, making it more difficult for a potential attacker to obtain any possible data.

The user himself is the most responsible for data security. How the user handle his credential is important in data security. If the user is aware, keeps his credential and device safe from any attacker then the data in the device is safe. But if the device is stolen or an attacker gets the credential for the application then it creates a security risk. RaiMobile uses the SQLite database to store data locally. Thus, if the phone is lost then it is possible for an attacker to get the application’s local database. And when the database is not encrypted then a data breach can occur. To avoid this worst-case scenario one of the possible solutions for database safety is to encrypt the database. Thus, even when the device is stolen the data stored in the device is safe. The only issue to this solution is the security of the key that is used to encrypt the database. Even if the database is encrypted, if the key is not kept safe then an attacker can decrypt it is using the key the then data is breached.

It does not matter how strong the door is if the key is not kept safe or not handled properly. Thus, it is the same with encryption. “The most important part of encryption is to keep the key used for encryption safe”. There are many ways how to keep the encryption key safe in an Android platform. Android Key-Store is a recommended way for keeping the encryption key safe in Android by Google. Android Key-Store generates a key for encryption, and it is stored in a secure hardware that makes it difficult for an attacker to extract from the device even if the user has root privilege.

2 DATA SECURITY

Protecting data from unauthorized use is data security. Data can include very basic information like name and age, as well as to important information like identity code, address, password's which is important information to an individual. Any breach to this can lead to serious damage in both business and to an individual. For an organization it can lead to the case where they must pay large fines to the government or the victim for not being able to protect user information. Data security is an important issue for a company to keep its business image safe and have customer trust. Any company makes a lot of investment for securing customer personal data. Even with all these investments, big companies are suffering from data breaches. For an individual it can lead to identity theft, loosing personal information or it can lead to personal attacks as well (OWASP, 2020).

2.1 Android Operating System

Android is an operating system for mobile devices. The core operating system is based on Linux kernel. Android is an open source platform and is owned by Google. It provides an application environment for mobile devices. Java and Kotlin are the primary programming languages used to write Android code. Android provides built in security features for Android applications. Android is designed to build mobile apps with a default system and file permissions. However, there are still data security issues related to storing or sharing the application data (Google, 2020).

2.2 Security Risk

Mobile technology is gaining popularity day by day. With the number of mobile applications growing every day, the data security risk for mobile applications increases. According to statistics, the most popular Android applications leak personal data. Of 250 popular Android mobile applications 70% leak sensitive personal data i.e. three out of every four applications (NowSecure, 2018). This leaves users data around the globe in danger of data breach, attack, identity theft and so on.

When an application is online then the data is stored in a server and an offline application stores the data in the user device locally. In this thesis project the main task is to secure the data in the user device. When data is stored in a user device it creates risk when the device is lost and when an attacker gets access to the device then it is a security concern. When the device is lost then the local data in it cannot be kept safe. Even though it is not easy, it is possible to get access to the local storage of the device and then if the device is rooted then it creates a security issue. Most security concerns for Android application are related to saving data on Android (OWASP, 2020).

2.2.1 Data Security In Android

As mobile devices are getting popular the importance of data security in mobile applications is also increasing. The Open Web Application Security project (OWASP) has identified the top risks for mobile applications. This includes risks and solutions for mobile application. If the data stored in the local storage are stored in plain text, then it is a risk. An attacker having physical access or using a malware can get the files (database, xml files and so on) stored in the device file system. If these files are stored as plain text, then the attacker can steal every information in these files for their benefits. OWASP categorise, insecure data storage, as the biggest threat in mobile application (OWASP, 2020).

Encryption is the solution for data security in Android. One of the major challenges of encryption is to protect the encryption key form an attacker. The cipher key which is needed to encrypt user data is again needed when there is a need for decrypting the data. Thus, for an offline application the key should be stored in the application local storage as it is not connected to the Internet and it is not possible to get the key from a server. Local storage is vulnerable to attack. It can be retrieved by an attacker with or without physical contact to the mobile device. Reverse engineering is possible for a mobile application and then an attacker can see the original mobile code and its pattern for obtaining the key (Kai, Q., Reza M. P.& Dan L. 2018). If the encryption key is stored as plain text or hardcoded in the application source file, then there is no point in encrypting the data. An attacker can decrypt it with the cipher key.

2.2.2 Android Cryptography

Java cryptography API provides classes and interface for the cryptographic functionality for Android. These classes provide functionality for data encryption, decryption, key generation and Message Authentication Code (MAC) generation. Android supports encryption such as symmetric, asymmetric, block and stream cipher. To keep this data safe, encryption is done (Google, 2020).

2.2.3 Android KeyStore System

Since the cipher key is critical for data safety Google Android team provides the Android KeyStore solution which allows to protect the cipher key. The Android KeyStore allows to generate a cipher key. The key generated by Android KeyStore is stored in the Android secure hardware of Android devices. The key material produced by Android key store never enters the application process. When the KeyStore feature is enabled, its key material is never exposed outside this secure hardware. It protects the key material from unauthorized use which prevents extraction of key material. The key generated by KeyStore can be protected by the screen lock. Whenever there is need for a key, the user must unlock the screen. Also, the key is an app specific meaning there is outside application restriction protecting the key from malware attack (Google, 2020).

With Android security library it is possible to add more security to protect a key stored in the KeyStore like user authentication every time there is a need to access to the key material in the Android KeyStore which adds more security to the key usage. Using Android security library, secret key material for encrypting encryption key can be generated. Key material is created at the first initialization of the application. Key material generated from Android KeyStore is safe and is stored in a secure hardware in the android system (Google, 2020).

KeyStore API includes parameters for setting the authentication. It is available above API 23. Thus, it is needed to set a lock for the key material usage to make it more secure. For this configuration, KeyStore calls two more parameters, *setUserAuthentucationRequired()* which is set to be true and *setUserAuthenticationValidityDurationSeconds()* which is set to some certain seconds. Now the application can retrieve the key material from the KeyStore within the set limit after unlocking the device. Android KeyStore uses device the screen lock of the device to protect the KeyStore key. If key material stored in KeyStore is not protected with a screen lock, then it creates a security weakness. It is possible to retrieve

reference to key material and then this could be used to initialize cipher to decrypt the essential key for decrypting the database (Pranczk. Fruba 2019). Thus, the key in the KeyStore should be protected by a screen lock.

KeyStore requires user authentication for its use. Whenever an application needs to retrieve or generate a new key material from KeyStore the user needs to authenticate with a screen lock or biometric lock. The Keyguard Manager API provided by Android, provides the method *isKeyGuardSecure* to check if a device has a screen lock or not. When the *isKeyGuardSecure* method is called it answer either *true* or *false*. If a screen lock exists, then access to keyStrore can be protected with it. If a screen lock exists, and the user has unlocked the device within a set time limit then the application can retrieve the key from KeyStore, otherwise the device needs to be unlocked again to retrieve the key. Once the user successfully authenticates himself with the screen lock then the key material can be retrieved within the time limit set in *setUserAuthenticationValidationDurationSec*. FIGURE 1. Android KeyStore initilization. presents the KeyStore initialization with a ten-second limit. Once the time limit is exceeded then it is needed to unlock the device again to get access to the key store. (Mobile-Security, 2020).

```
javax.crypto.KeyGenerator keyGenerator =
        javax.crypto.KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES, Const.ANDROID_KEY_STORE);
keyGenerator.init(new KeyGenParameterSpec.Builder(Const.KEY_ALIAS,
        purposes: KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT).setBlockModes(
        KeyProperties.BLOCK_MODE_GCM)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
        .setUserAuthenticationRequired(true)
        .setUserAuthenticationValidityDurationSeconds(10)
        .build());
return new SecurityKey(keyGenerator.generateKey());
```

FIGURE 1. Android KeyStore initilization.

2.2.4 Forgetful KeyStore

KeyStore protects key material using the screen lock of a device. But there is a case when the screen lock can be removed, the lock type can be changed or in the worst-case the screen lock can be removed. When a parameter of KeyStore instance *setUserAuthenticationRequired* is set to *true*, then all the key material stored in KeyStore would be lost if the screen lock is removed with no possibility of retrieving. Meaning the key material that the application uses to encrypt or decrypt its information is deleted. This feature of KeyStore is done so by design to make it more secure (Doridori, 2020). Thus, to prevent this case from occurring it is requirement to not remove screenlock once the application is in use or to has data backup.

2.2.5 Encryption

Encryption is the process of encoding information. By encrypting, information can be converted from simple text to code or symbols also known as cipher text. Encryption is done to protect the information from unauthorized check. Plain text is encrypted to cipher text with an algorithm and a generated key. Converted cipher text can only be decrypted using the key which was used during decryption. The Cipher class provides functionality of cryptographic cipher for encryption and decryption in Android. The Cipher object is created using the *getInstance* method from Cipher class. It accepts a specified transformation in the form of algorithm/mode/padding for the encryption. Then cipher is initialized with encrypt mode in this case, a cipher key and algorithm parameter. When cipher initialization text is to be encrypted it is passed to be encrypted using the *doFinal* method of cipher instance which returns the encrypted data in byte format. Here the data is encrypted in a single part operation as shown in FIGURE 2. Encryption The transformation and algorithm used in this project are recommended by Google Android team (Oracle, 2020).

```
try {
    Cipher cipher;
    cipher = Cipher.getInstance("AES/GCM/NoPadding");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey,
            new GCMParameterSpec( tLen: 128, "AES/GCM/NoPadding".getBytes(), offset: 0, len: 12));
    byte[] encrypted = cipher.doFinal(token.getBytes());
    return Base64.encodeToString(encrypted, Base64.URL_SAFE);
} catch (GeneralSecurityException e) {
}
```

FIGURE 2. Encryption

2.2.6 Decryption

Decryption is a process of decoding encrypted information. Information encoded to symbol and code by encryption is turned back to the original text by decryption. Decryption is done with the same key and algorithm which were used when the information was encrypted. Like that of the encryption *Cipher* class provides functionality of cryptographic cipher for decryption in Android. A *Cipher* object is created using the *getInstance*() method from *Cipher* class. It accepts a specified transformation in the form of algorithm/mode/padding for encryption. Then cipher is initialized with the mode decrypt in this case, the cipher key and algorithm parameter. Encrypted text is first decoded into byte and decrypted by passing to the *doFinal*() method of cipher instance which return decrypted original text as shown in FIGURE 3. Decryption The transformation and algorithm used in this project are the same as in encryption (Oracle, 2020).

```
try {
    Cipher cipher;
    cipher = Cipher.getInstance("AES/GCM/NoPadding");
    cipher.init(Cipher.DECRYPT_MODE, secretKey,
            new GCMParameterSpec( tLen: 128, "AES/GCM/NoPadding".getBytes(), offset: 0, len: 12));
    byte[] decoded = Base64.decode(encryptedToken, Base64.URL_SAFE);
    byte[] original = cipher.doFinal(decoded);
    return new String(original);
} catch (GeneralSecurityException e) {
}
```

FIGURE 3. Decryption

2.3 Security of RaiMobile

RaiMobile is a mobile application owned by Raisoft. It is an offline mobile solution of the Raisoft web application. Java programming language is used to develop RaiMobile. RaiMobile is a dual offline/online application. RaiMobile uses the SQLite database to store information of the user. It uses local shared preference to store key value data like server tokens, credentials and so on. Thus, the data security of user of RaiMobile is necessary. Mobile security includes many risks. It is a too large topic for a single thesis. In this thesis, the focus is on the security of the local storage.

2.3.1 Risk Analysis

RaiMobile stores multiple user information that includes identity number, address, medical condition and so on, which can lead to an attack or privacy violation for the client. RaiMobile also stores tokens for server connection. Credential theft can lead to identity theft. Currently users' credential and user data is stored in simple text in a local storage. If the credential is leaked, then an attacker can pretend to be the user of the application and steal data that they have access to. RaiMobile uses SQLite database to store information. Data that are private to the user are stored in the database. The database is stored in an application local storage. In the current solution the database is not encrypted. Thus, there are multiple threats to the current database. Malware attacks and database theft are possible. If an attacker manages to get the database, then all the information is stolen.

2.3.2 RaiMobile data security

RaiMobile should protect the user data that is stored a local storage. In the local storage it is possible to store multiple users' information. This information includes the users' medical condition as well as some personal information. RaiMobile stores also access and refresh tokens which are stored when the user logs in to the Raisoft server. Using tokens the user can be logged in to the server for a long period of time. It will also store the encryption key for encrypting the database after the current project implementation. The encryption key is the top information that should be protected. If it is leaked, then there is a possibility to lose other data along with it. Since all the other data are stored in the database this information are to be protected.

3 APPLICATION DESIGN

This chapter contains all the design requirements needed for the application development. Application design includes requirement engineering, application use case, identifying class based on use cases, application flow and state diagram. The final application will be based on this design.

3.1 Application Idea

The main aim of this thesis is to create a solution for RaiMobile to solve its local storage security issue. RaiSecure is an application made as proof of concept to solve the local data security issue related to RaiMobile. RaiSecure has a local SQLite database to store dummy information. The database is to be encrypted in users' first login with the key it gets from a mock server. The key is only received after a successful login. When a user is successfully logged in to the application, a server request is made which in return sends the encryption key. For server login the internet is needed. The key provided by server is used to encrypt the user database. RaiSecure supports offline/online login. During the first login the user sets a pin for offline login. The Pin can be changed or recreated if forgotten by logging in to the server.

During the first login the database is created and encrypted using the cipher key provided by the server. If a new user logs in, then a new database is created and encrypted using a new cipher key provided by the server. The key that the application receives is used to encrypt the new created user database. After encryption of the database, the key is encrypted using key material provided by Android KeyStore. For Android KeyStore to provide a key the user needs to unlock the device using a screen lock. After encrypting key, it is stored in Encrypted Shared Preference. The application uses hybrid shared preference for keeping data safe (Sourabh, S, K., Young, L. & Jeong Y. 2020)

If user already exist, then the a database needs to be decrypted to use the application. After a successful user login, the user needs to unlock the application to get key material from KeyStore. The key material is used to decrypt the key used for database decryption. After database decryption , the user can write and read from the database. To handle correct flow, follow the best practise and make the solution compatible with the bigger project RaiMobile is the main goal of this thesis.

3.2 Requirements

Requirements for a system are a collection of what the system should do, what the system should not do and its usage. Doing requirements documentation of a system to be developed is called requirements engineering. It includes research, analysing, documentation of requirements for system. Requirement engineering helps to understand customer needs. Requirement engineering is done to meet the needs of customers for the system based on risk analysis (Sommerville, 2011).

3.2.1 Functional Requirement

Functional requirement states what the system should do. It describes the functionality of the system. Its requirement depends on the type of software, expected users and the system where it is used (Sommerville, 2011). RaiSecure should satisfy the following functional requirements:

FR001 Application shall be able to display user notes.
FR002 User shall be able to set pin after login.
FR003 Application shall allow user to change the application pin.
FR004 Application shall encrypt the database with a cipher key that it gets from server after login.
FR005 User shall set up pin with at least 4 digits.

3.2.2 Non-Functional Requirement

Non-functional requirement describes how the system is supposed to work. It is a set of standards to measure the application. It includes important requirements like security requirement, storage requirement and so on (Sommerville, 2011). RaiSecure has the following non-functional requirements:

NFR01 Application should have access to internet only when login.
NFR02 Application should be able to work offline.
NFR03 Application should validate user credential.
NFR04 Application should make HTTP request for credentials.
NFR05 Application should encrypt database.

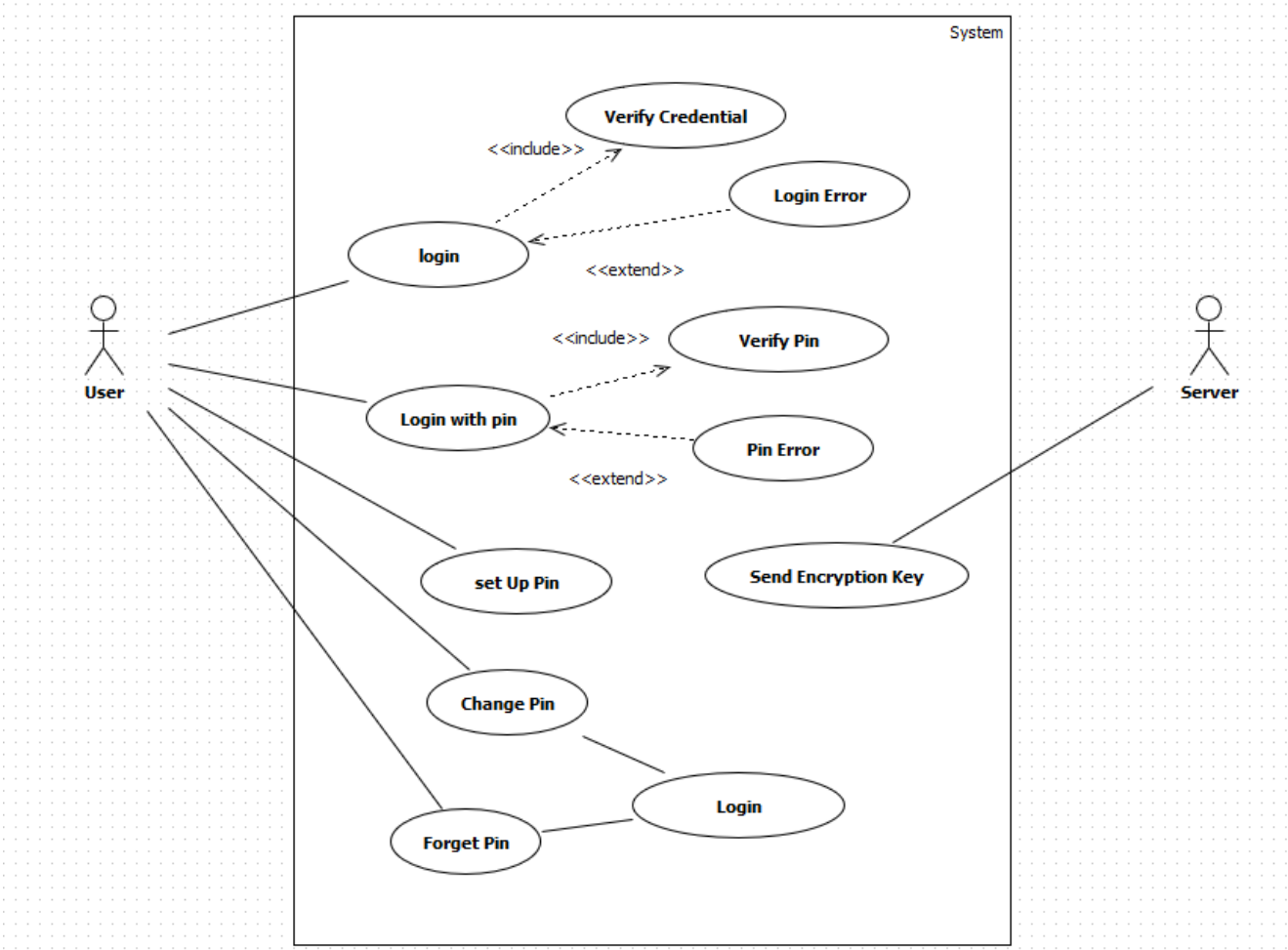
3.2.3 System requirements

System requirement is the requirement of the system that the software is to be run. It applies to the system as a whole and is not the requirement to run for certain functionality (Sommerville, 2011). RaiSecure has the following system requirements to be met to be able to use it:

SR001	The application shall work on platform above Android 6.0.
SR002	Application should have a screen lock.

3.3 Use case

Use case is a simple description of uses of a system or an application. A use case describes the interaction that an actor does with the system. An actor can be a user, server, local storage and so on. It demonstrates what action needs to be done to reach the intended task. It documents the scope of the system. Use case is needed to demonstrate how the system will work in user events. (Sommerville, 2011)



GRAPH 1. Use case Diagram

3.3.1 Use Cases Description

The use case description contains information about the single use case event. With the analysis of the use case description class attributes and operations can be derived. Use case description includes information of use case event like preconditions, normal flow, trigger of use case and so on.

Use Case ID:	001
Use Case Name:	User Login
Actors:	User
Description:	Use case describes the process of login
Trigger:	User

Pre-conditions:	User uses the application for first time
Normal Flow:	User input username and password in the input field. User clicks the login button. Application validates user credential [User not Found]. Application shows set pin view.
Alternative Flows:	
Exceptions:	[User Not Found]: Error user does not exist.
Post-conditions:	On success, user move to next view to set up application pin. Otherwise user NOT LOGGED in
Includes:	Validate user credential.
Frequency of Use:	Every time when user is not logged in.

TABLE 1. Login Use Case

Use Case ID:	002
Use Case Name:	Login with PIN code
Actors:	User
Description:	User login with PIN code.
Trigger:	User
Pre-conditions:	Application is logged in.
Normal Flow:	Use enter PIN. Application validate the pin code [Incorrect PIN]. Application navigate to notes view.
Alternative Flows:	
Exceptions:	Incorrect PIN: PIN does not match.
Post-conditions:	On success, user move to next view notes.
Includes:	Verify user PIN.
Frequency of Use:	Every login

TABLE 2. Login with Pin Code Use case

Use Case ID:	003
Use Case Name:	Send encryption Key

Actors:	Server
Description:	Sever send encryption Key
Trigger:	User
Pre-conditions:	User login the application for first time
Normal Flow:	Server send application encryption key after it receive get request.
Frequency of Use:	Every new login

TABLE 3. Send Encryption Key Use Case

Use Case ID:	004
Use Case Name:	Set up PIN
Actors:	User
Description:	User set up PIN code.
Trigger:	User
Pre-conditions:	User logged in to server
Normal Flow:	User successfully log in to the system. User set up pin [PIN ERROR] and confirm pin again [PIN not match]
Alternative Flows:	
Exceptions:	PIN ERROR: PIN should be at least 4 digits. PIN does not match: PIN does not match the previous pin.
Post-conditions:	On success, user move to next view notes.
Includes:	Verify user PIN.
Frequency of Use:	Every new login

TABLE 4. Set Up Pin Use Case

Use Case ID:	005
Use Case Name:	Forgot PIN
Actors:	User
Description:	User forget pin and change Pin to login in application
Trigger:	User
Pre-conditions:	User is online

Normal Flow:	User forget then pin. For user to login user have to login into server to set up new PIN. User click forget button and navigate to login view.
Alternative Flows:	
Post-conditions:	On success, user move to next view to set up application PIN
Frequency of Use:	Only when user want to change application pin or forget PIN

TABLE 5. Forget PIN Use Case

3.4 Class Selection

Class Diagram used in applications can be derived from its use case. The class is a collection of attributes that describe its properties. Class attributes can be derived from the use case. The use case gives the description of actor action on the system. Thus, with the information provided in the use case description class elements can be derived (Sommerville, 2011).

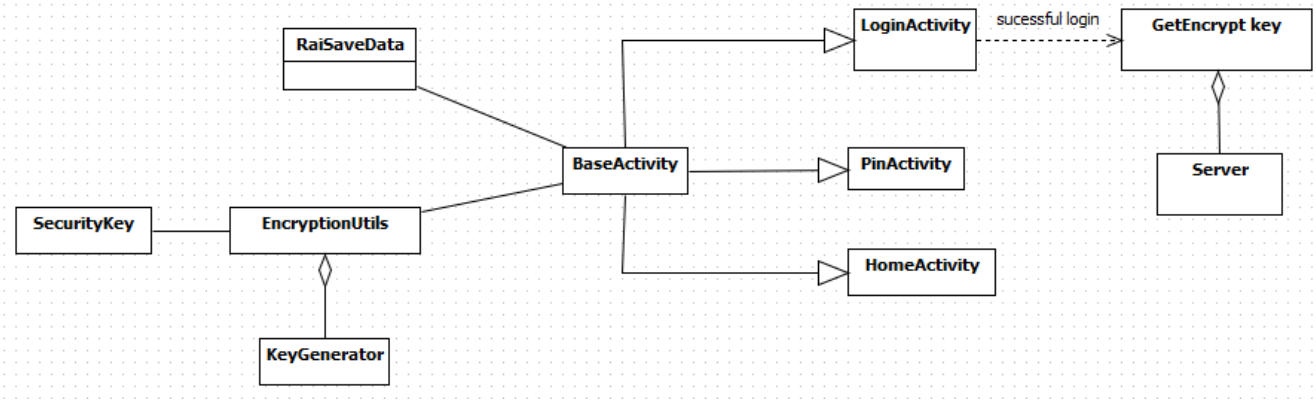
Use case number	Use case name	Use case description
001	User Login	User input username and password in the input field. User clicks the login button. Application Validate user credential [User not Found]. Application shows set pin view.
002	Login with PIN code	Use enter PIN. Application validates the PIN code [Incorrect PIN]. If PIN is greater than four digits. Application navigate home view.
003	Get Encryption Key	Server send application encryption key after it receive get request from RaiMobile.
004	Set Up PIN	User successfully log in to the system. User set up pin [PIN ERROR] and confirm PIN again [PIN not match]
005	Forget PIN	User forget then PIN. For user to login user have to login into server to set up new PIN. User click forget button and navigate to login view.

TABLE 6. Lexical analysis of requirement

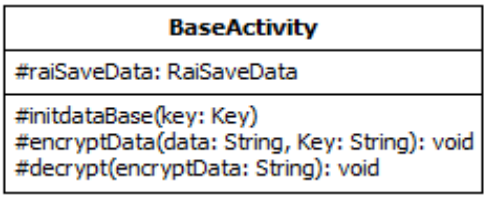
From the analysis of TABLE 6. Lexical analysis of requirement following class candidates were selected: username, password, login button, PIN view, encryption key, PIN, home view, forget button.

3.5 Class Diagram

These selected items are the possible classes, attributes of class. With use case, the result of user interface and data model class attributes can be categorised and differentiated. Common attributes are combined for a single class. An application will have three view classes. The Application has a database but it is an empty database. Sole purpose of this database is to encrypt it. The class diagram in GRAPH 2. Class Diagram demonstrate abstract UML class diagram for the class used in the application.



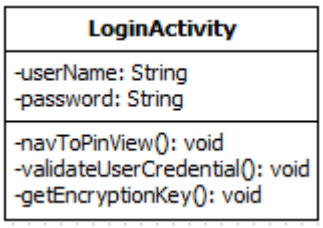
GRAPH 2. Class Diagram



GRAPH 3. BaseActivity

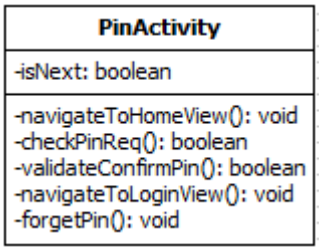
BaseActivity is an activity class. It is the parent class to all other view classes. All the other view classes inherit from *BaseActivity* activity class. Creating the base class will prevent duplicate methods in view class and reduce boilerplate codes. Once a user is successfully logged in, app sends a request to the

server for an encryption key. It is called(*initDatabase*) when the application gets key from server. GRAPH 3. BaseActivity 3 show the Base activity class diagram.



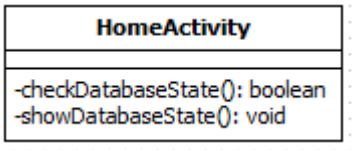
GRAPH 4. LoginActivity

LoginActivity shown in GRAPH 4. LoginActivity the login view class of application. It has plain text views for username and password respectively. The login button in the view is used to log in to the application. The login button has click event to validate user credential. If the credentials are correct it sends a GET request to the server to get the encryption key. The local database is encrypted with the key received from the server. The key received is encrypted and stored in shared preferences.



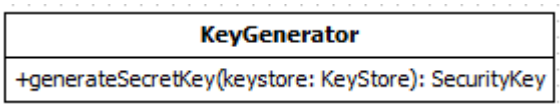
GRAPH 5. PinActivity

PinActivity shown in GRAPH 5. PinActivity view class is used for setting up a new pin or for pin login. On this view a numeric keyboard is shown and the input field view for the pin. A floating action button is clicked when the user want to confirm the pin or move to the next view. *isNext* attribute is true when user enter PIN for first time and press next button to confirm PIN.



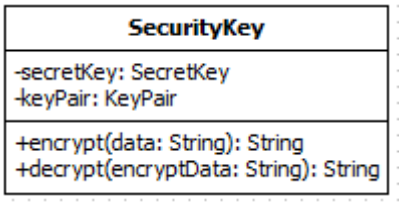
GRAPH 6. HomeActivity

HomeActivity as shown in GRAPH 6. HomeActivity displays single text view. This text view displays whether the database is successfully encrypted or not. If the database is successfully loaded, then the database state is checked whether it is corrupt or not. The result of the database state is displayed to verify a successful encryption of the database.



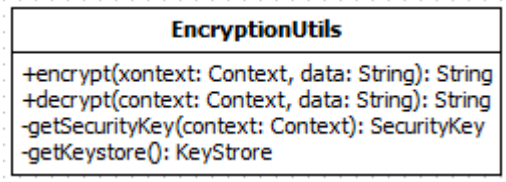
GRAPH 7. KeyGenerator

KeyGenerator presented in GRAPH 7. KeyGenerator is used to generate key material from the KeyStore. The *KeyGenerator* contains a single method and it returns a *SecurityKey* object. If the application is run for first time, then it generates a new key and returns it, if not it returns the old generated key.



GRAPH 8. SecurityKey

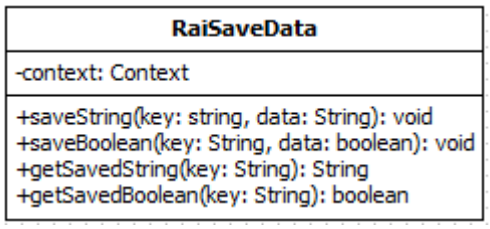
SecurityKey shown in GRAPH 8. SecurityKey class is a model class used to store a secret key or key pair data generated by KeyStore. It is a placeholder class and holds two data pairs. It also has two methods *encrypt* and *decrypt*. These methods encrypt and decrypt give a data pair and return them respectively.



GRAPH 9. EncryptionUtils

EncryptionUtils presented in GRAPH 9. EncryptionUtils class contains the operation for encryption, decryption and getting key material from the Android keyStore. It includes two dependency classes *SecurityKey* and *KeyGenerator*. It links both classes to provide support for encryption and decryption.

The main view class has an instance of *EncryptionUtils* class with which data encryption and decryption is done by other inherited view classes.

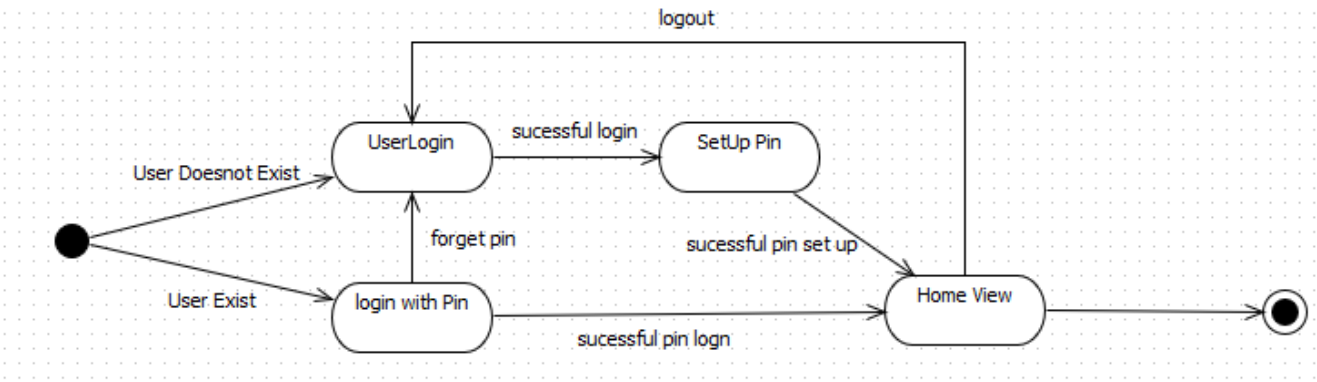


GRAPH 10. RaiSaveData

RaiSaveData in GRAPH 10. RaiSaveData is a class used for saving data in local resources. An encrypted shared preference is initialized in this class using context provided in the constructor. Using the instance of encrypted shared preference data is saved to local XML along with value key. Data stored in shared preference can be retrieved using the key used when storing values.

3.6 Activity Diagram

An activity diagram is a behavioural diagram. It provides a graphical representation of the flow of the system. It describes how an application works from start to finish (Sommerville, 2011). The application RaiSecure has in total three views. But the navigation flow always differs if the user is logged in or not. If the user is not logged in, the user needs to log in to the server then create a PIN to enter the app process. If the user exists, then the user can enter the app process by entering the PIN code which is created in the first login. If the user forgets the PIN, then the user creates a new PIN by logging in to the server. Thus, basically it follows the same flow when the user uses app for the first time. GRAPH 11. Activity Diagram illustrates the activity diagram for RaiSecure.



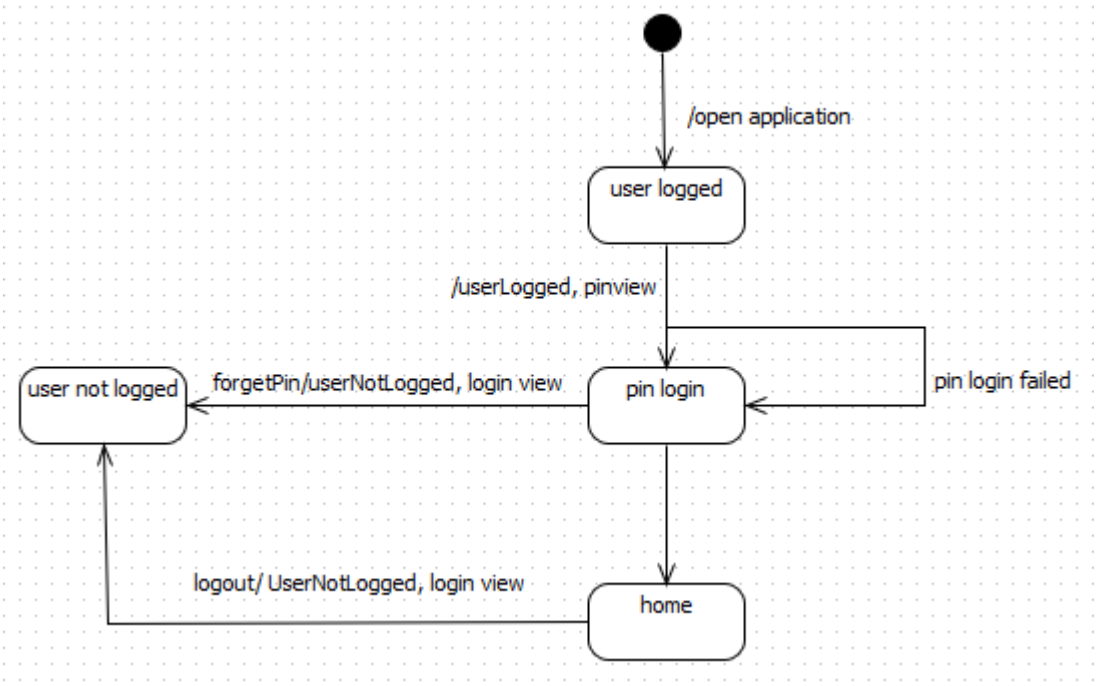
GRAPH 11. Activity Diagram

3.7 State Diagram

A logged user and a not logged user are the two states of user in RaiSecure. When a user is successfully logged in to the server then the user is logged. If no user exists in RaiSecure then the state is not logged. The logged and not logged user flow differ from one another. A logged user can simply enter the app process by using a PIN or doing the server login whereas a not logged user needs to log in to the server and set up an application PIN to enter the application process.

3.7.1 Logged User

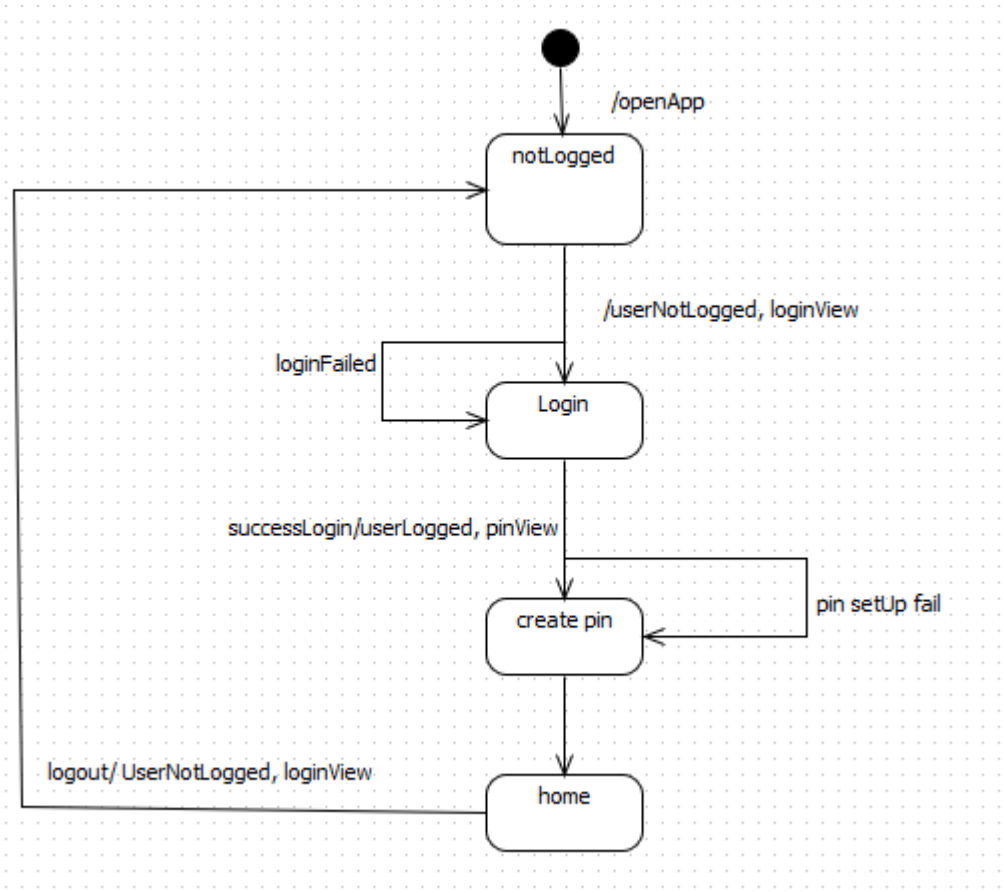
When a user is not logged, GRAPH 12. State diagram Logged User shows the change in state of the user from logged to not logged. It shows the state of logged user when using the application. If the user forgets the PIN or does log out then the user state changes from logged to user not logged.



GRAPH 12. State diagram Logged User

3.7.2 Not Logged User

When a user is not logged then the only way for the user state to be logged is to login to the application server. When the user logs in to the server then the user state changes from not logged to logged. GRAPH 13. State diagram not logged user shows the state diagram of the user when the user is not logged.



GRAPH 13. State diagram not logged user

3.8 User Interface

RaiSecure has three view screens, the login view for the user to login to the server, the pin view for the user to setting up a pin, and the home view. The home view contains a simple text box which notifies if the database is successfully encrypted or not. When a user logs in to the application by creating an account (APPENDIX 1), the system will send one get request to the server. The response for this request which the application gets contains the encryption key which is used to encrypt the database. After logging in the user sets up a pin for offline and faster login (APPENDIX 2). The user sets up a pin code. After setting up the pin code the user can use the application features. When the user uses the application again then the user can do pin login (APPENDIX 3).

3.9 Application Development

This chapter contains information about design, external libraries, development tools and more on how the encryption of the database and key handling is done. It also contains a few code snippets of important encryption logic.

3.9.1 Development Tools

Android Studio 3.6 was used for the development of this application. Android Studio is an official development environment for Android applications. It was developed by Google and JetBrain. Android Studio provides backward compatibility for application to run on lower API level (Google, 2020). All library provided by Android are migrated to AndroidX extension to have the latest feature available in development.

3.9.2 Retrofit

Retrofit is a REST Client library. It is used to create HTTP request and process HTTP response from REST API in Android. It is created and maintained by Square. RaiSecure uses Retrofit to make HTTP requests to get encryption keys as response to encrypt the database. REST sends encryption key in JSON format. Retrofit provides Gson support to convert Java Objects to JSON representation and vice versa respectively (Square, 2020).

3.9.3 DBFlow

DBFlow is an SQLite library for Android. It is built with annotation processing thus it generates all the necessary code needed for database creation with less code. It is easier, faster and an application written with it is less error prone. It is created and developed by Raizlabs. DBFlow also provides support encryption using SQLCipher for database Encryption (Agrosner 2020).

3.9.4 SQLCipher

SQLCipher is an open source extension to the SQLite database. SQLCiper is developed and maintained by Zetetic. It is a widely used library for encrypting database in Android. It uses 256-bit AES encryption algorithm (Zetetic.net. 2020). Since DBFlow replaces most of SQLCipher interaction with its own implementation the encryption of database becomes simpler. First, the database is initialized using DBFlow and then later encrypted by using the encryption key. Thus, the initialization of the database always happens after user login. In FIGURE 4. SQLite Encryption the initialization of database and its encryption is shown.

```
public void initDatabase() {
    FlowManager.init(FlowConfig.builder(getApplicationContext())
            .addDatabaseConfig(DatabaseConfig.builder(SecureDatabase.class)
                    .databaseName(getDatabaseName())
                    .openHelper((databaseDefinition, helperListener) -> {
                        SQLCipherHelperImpl sqlCipherHelper = new SQLCipherHelperImpl(databaseDefinition, helperListener);
                        sqlCipherHelper.setKey(getEncryptKey());
                        return sqlCipherHelper;
                    })
                    .build())
            .build());
}
```

FIGURE 4. SQLite Encryption

3.9.5 AndroidX Security

Android Security library is a part Android Jetpack. This library enables to use wider features of Android KeyStore for key generation and Encrypted Shared Preference which is used for encrypting local shared preference. This library guides to work with best security practice that is followed for data protection. Although it is part of Android Jetpack it is already a stable and available as alpha library (Google, 2020).

3.9.6 Conductor

Conductor library is a framework that allows building view-based application instead of fragments provided by Google. Conductor has a simpler lifecycle compared to that of a fragment. *Controller*, *Router*, *ControllerChangeHandler* and *RouterTransation* are components of controller that make the use of this library easier (Bluelines, 2020).

4 APPLICATION TEST

In order to ensure proper application behavior, certain tests are carried out. Since the application was small with only a few screens, most of UI the tests were done manually. But in this application the most crucial test is needed for encryption. Tests were done trying to break the encrypted database and decrypting the key stored in shared preference. Also, the critical flow of the application is tested as well. Since a screen lock cannot be removed when RaiSecure is installed its case is not tested. Test result are presented below in TABLE 7. Test Result. Each test case contains the test action, expected behaviour to the test action and the result of the test. Success represents the success of the test and if some malfunction occurred the test result is written as failure.

Action	**Expected Behaviour**	**Test Result**
User changes screen lock after setting up RaiSecure. User use RaiSecure again.	Successful database decryption.	Success
User changes screen lock type from current type to PIN, passcode or pattern or vice versa after setting up RaiSecure. User use RaiSecure again.	Successful database decryption.	Success
User uses application without setting up PIN.	Application display PIN set up view as first screen.	Success
User uses application offline after setting up application PIN.	User successfully login using PIN and use application.	Success
Opening encrypted database without key.	Database should not be able to open without encryption key.	Success
Opening encrypted database with correct key	Database is opened.	Success

Encrypted xml file is checked	Both key value pair are encrypted	success
New user login	New encrypted database is created with username.	success
N user login to RaiSecure	N encrypted database is created with username	success

TABLE 7. Test Result

5 CONCLUSION

The main aim of this thesis was to provide proof of concept solution to a larger project RaiMobile. RaiSecure created during this thesis tries to tackle the security problem faced by RaiMobile in storing local data. Data security is important and gaining importance over time. This application was developed from scratch but with the help of open source library.

Of the many topic available, the reason to choose this topic was because of the growing importance of data security in mobile application. The development process took about three weeks of development. The design solution used in this solution follows best security practise in the Android community. Security in Android has evolved and with Android Security library additional feature for data security are added.

Local storage is not only security issue for mobile application. There are many risks for data security but for offline application it is the most important one to be resolved. This approach needs more investigation and fixing of security vulnerability. Fixing the overall security flow is too large topic for a single thesis project and needs a lot of investigation and planning.

In recent years mobile applications has grown fast. Now more and more web applications are migration to mobile application as mobile (smart phones) are gaining popularity day by day. A mobile application contains a lot of private information about the user. Data security importance grows with the number of mobile users growing and mobiles containing more and more user private information.

REFERENCES

Agrosner. 2020. Available: https://github.com/agrosner/DBFlow. Accessed 1 March 2020.

Blueline. 2020. Available: https://github.com/bluelinelabs/Conductor. Accessed 15 March 2020.

Doridori 2015. Available: https://doridori.github.io/android-security-the-forgetful-keystore/. Accessed 12 April 2020.

Google. 2020. Available: https://developer.android.com/. Accessed 20 March 2020.

Square. 2020. Available: https://square.github.io/retrofit/. Accessed 20 March 2020.

Kai, Q., Reza M. P.& Dan L. 2018. OWASP Risk Analysis Driven Security Requirements Specification for Secure Android Mobile Software Development. Available: https://ieeexplore-ieee-org.ezproxy.centria.fi/stamp/stamp.jsp?tp=&arnumber=8625114. Accessed 8 March 2020.

K. Pranczk & M. Fruba 2019. How secure is your Android KeyStore authentication? Available: https://labs.f-secure.com/blog/how-secure-is-your-android-keystore-authentication/. Accessed 12 April 2020

Mobile-Security. 2020. Data Storage on Android. Available: https://mobile-security.gitbook.io/mobile-security-testing-guide/android-testing-guide/0x05d-testing-data-storage. Accessed 4 March 2020.

Nowsecure. 2019. INFOGRAPHIC: Test of 250 Popular Android Mobile Apps Reveals that 70% Leak Sensitive Personal Data. Available: https://www.nowsecure.com/resource/infographic-test-of-250-popular-android-mobile-apps-reveal-that-70-leak-sensitive-personal-data/. Accessed 10 April 2020

Oracle. 2020. Available: https://docs.oracle.com/javase/7/docs/api/javax/crypto/Cipher.html. Accessed 25 March 2020.

OWASP. 2020. OWASP Mobile Top 10. Available: https://owasp.org/www-project-mobile-top-10/. Accessed 4 April 2020

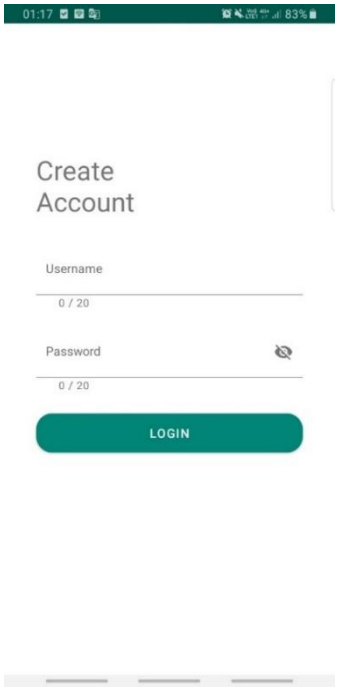
Raisoft. 2020. Available: https://www.raisoft.com/. Accessed 15 April 2020

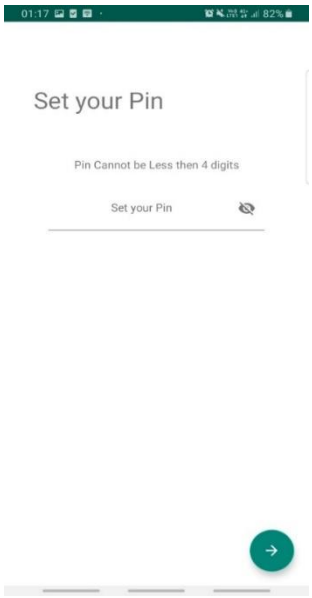
Sourabh, S, K., Young, L. & Jeong Y. 2018. Hybrid Encryption for Securing Shared Preferences of Android Application. Available: https://ieeexplore-ieee-org.ezproxy.centria.fi/stamp/stamp.jsp?tp=&arnumber=8367771. Accessed 14 March 2020

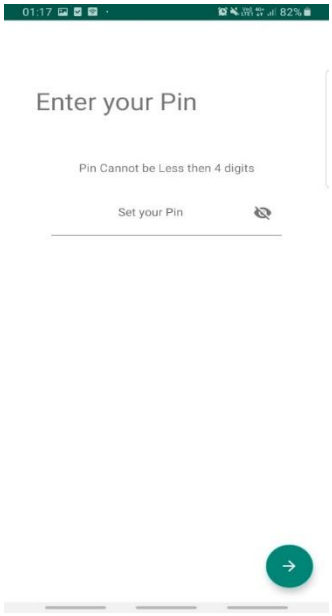
Sommerville, I. 2011. Software Engineering

Zetetic.net. 2020. Available: https://www.zetetic.net/sqlcipher/. Accessed 8 March 2020

APPENDIX 1




01:17
82%
Set your Pin
Pin Cannot be Less then 4 digits
Set your Pin


01:17
82%
Enter your Pin
Pin Cannot be Less then 4 digits
Set your Pin